ThoughtWorks®

TECHNOLOGY RADAR VOL. 21

有态度的前沿技术解析

thoughtworks.com/radar
#TWTechRadar

技术雷达中国区技术咨询顾问组

魏喆/吴瑞峰/边晓琳/李光毅/方明/樊卓文/刘先宁/徐瑾/张霄翀/梁凌锐/马伟/张刚/尹尚维/徐培/万学凡/陈孟/张丽/闵锐/黄进军/阮焕/陈璐/喻晗/张扬/易维利/伍斌/包欢/王瑞鹏/姚琪琳/邬倩/韩盼盼/张凯峰/林从羽/王家骥

贡献者

技术雷达由ThoughtWorks技术顾问委员会筹备，其人员组成为：


Rebecca Parsons (CTO)


Martin Fowler (Chief Scientist)


Bharani Subramaniam


Erik Dörnenburg


Evan Bottcher


Fausto de la Torre


Hao Xu


Ian Cartwright


James Lewis


Jonny LeRoy


Ketan Padegaonkar


Lakshminarasimhan Sudarshan


Marco Valtas


Mike Mason


Neal Ford


Ni Wang


Rachel Laycock


Scott Shaw


Shangqi Liu


Zhamak Dehghani

本期技术雷达是基于ThoughtWorks技术顾问委员会在2019年10月旧金山会议上的讨论所得出的。

关于
技术雷达

ThoughtWorker酷爱技术。我们对技术进行构建、研究、测试、开源、描述，并始终致力于对其进行改进，以求造福大众。我们的使命是支持卓越软件并掀起IT革命。我们创建并分享ThoughtWorks技术雷达就是为了支持这一使命。由ThoughtWorks中一群资深技术领导组成的ThoughtWorks技术顾问委员会创建了该雷达。他们定期开会讨论ThoughtWorks的全球技术战略以及和对行业有重大影响的技术趋势。

这个雷达以独特的形式记录技术顾问委员会的讨论结果，从首席技术官到开发人员，雷达为各路利益相关方提供价值。这些内容只是简要的总结，我们建议你探究这些技术以了解更多细节。这个雷达的本质是图形性质，把各种技术项目归类为技术、工具、平台和语言及框架。如果雷达技术可以被归类到多个象限，我们选择看起来最合适的一个。我们还进一步将这些技术分为四个环以反映我们目前对其的态度。

想要了解更多技术雷达相关信息，请点击：thoughtworks.com/cn/radar/faq

雷达一览

1 采纳

我们强烈主张业界采用这些技术。我们会在适当时候将其用于我们的项目。

2 试验

值得追求。重要的是理解如何建立这种能力。企业应该在风险可控的项目中尝试此技术。

3 评估

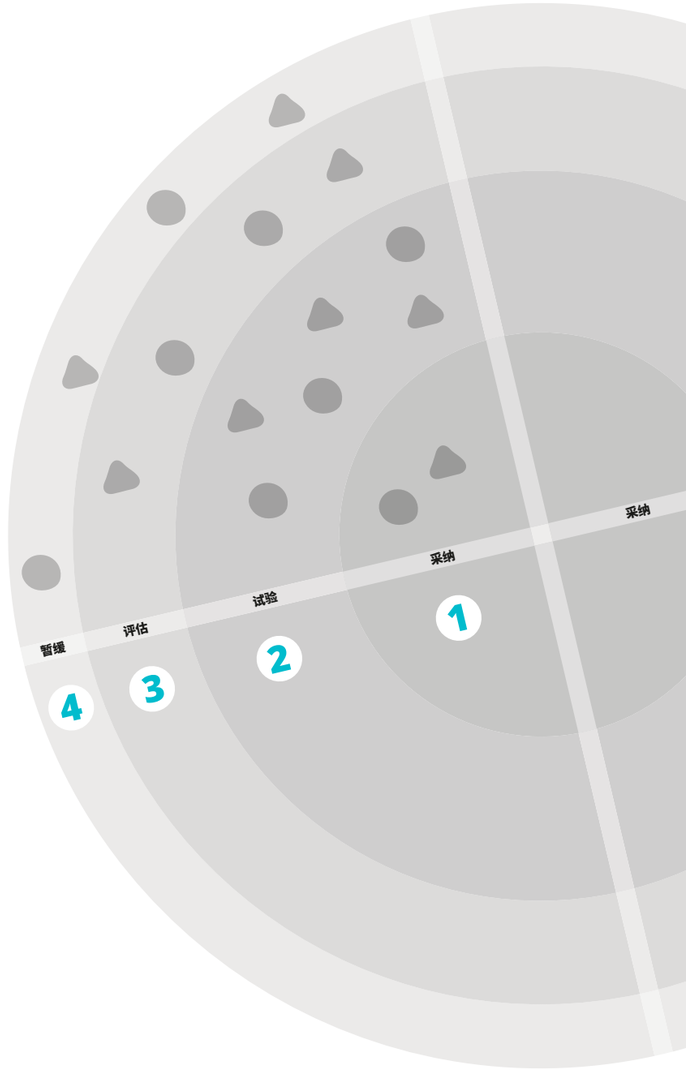
为了确认它将如何影响你所在的企业，值得作一番探究。

4 暂缓

谨慎推行

▲ 新的或发生变化的
● 没有变化

! 技术雷达是具有前瞻性的。为了给新的技术条目腾出空间，我们挪走了近期没有发生太多变化的技术条目，但略去某项技术并不表示我们不再关心它。



最新动态

本期精彩集锦

云：多即是少？

在主流云服务提供商提供的核心功能日益趋同的当下，竞争焦点已经转移到他们能够提供的附加服务上，这就鼓励了云服务商以惊人的速度发布新产品。为在竞争中取得优势，很多新服务还在存有瑕疵、功能尚不完整的阶段，就被匆匆推向市场。过分关注速度以及产品扩张，常常导致这些由收购或仓促打造而来的服务缺陷频出，文档质量差，难于实现自动化，以及与服务商自己的其他服务不能完整集成。团队在尝试使用云服务提供商承诺的功能却经常遇到问题的时候，会感到挫败。考虑到在选择云服务提供商时，通常都是由组织高层基于各个不同维度进行决策，我们建议实施团队：不要假设你们的云服务提供商的所有服务都能确保相同的质量，应该对关键功能进行测试。如果你们的产品上市时间能够容纳额外的运维开销，可以考虑替代的开源解决方案，或者使用多云策略。

保护软件供应链

组织应该抵制冗长的人工检测和需要审批的象牙塔治理规则。相反，自动化的依赖保护（依赖漂移适应度函数），安全性（安全策略即代码）和其他治理机制（运行成本纳入架构适应度函数）可以保护软件项目中重要但不紧急的部分。这个关于策略、合规性和治理即代码的主题在我们对话中多次出现。我们看到自动化不断提升的软件开发生态系统的自然演化：与自动测试、持续交付和基础设施即代码的持续集成，以及如今的自动化治理。云成本、依赖管理、架构结构等以往人工流程的自动化呈现出一种自然演进的态势；我们正在学习如何让软件交付中的所有重要方面自动化。

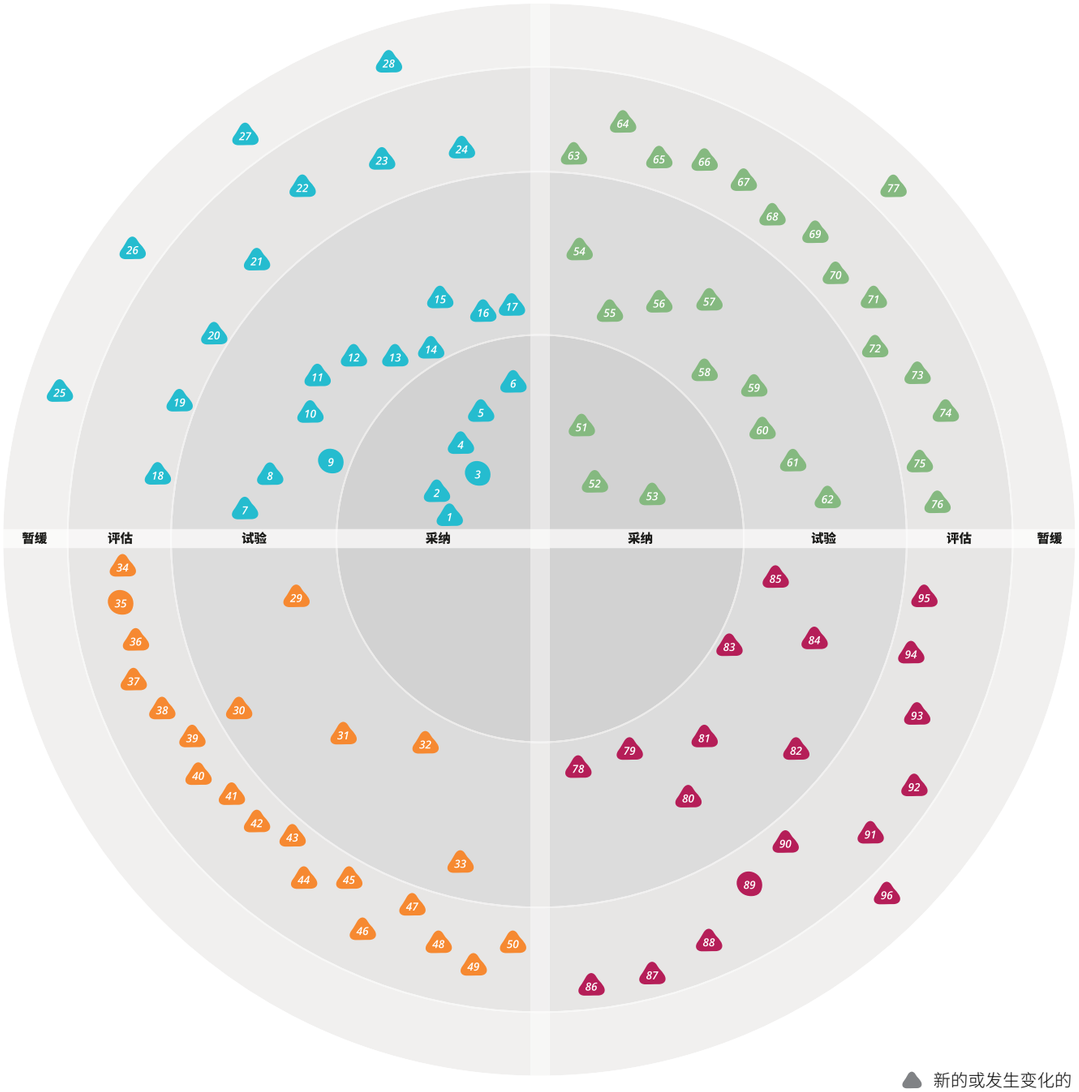
打开机器学习的黑匣子

通过使用模式匹配、反向传播和其他众所周知的技术，机器学习通常貌似能解决人类所无法解决的问题。然而，尽管这些技术功能强大，但许多模型本质上令人费解。这意味着机器学习所计算的结果，无法用逻辑推理来解释。所以，当需要了解决定是如何做出，或存在将偏见、抽样、算法或其他偏差引入模型的风险时，人们就会一筹莫展。现在我们看到，诸如假设分析之类的工具，以及道德偏差测试这样的技术正在涌现出来。这些工具和技术可以帮助我们发现模型的局限性，并预测模型的输出结果。尽管这些在可解释性方面的改进，是朝着正确方向迈出的一步，但理解深度神经网络，仍然是一个遥不可及的目标。因此，在选择机器学习模型时，数据科学家开始将可解释性视为第一要务。

软件开发是一项团队运动

技术雷达诞生之初，我们就提醒不要使用那些会分隔开发团队、阻碍反馈及协作的工具和技术。通常，当新的专业人才参与项目后，为避免陷入“常规”开发的混乱局面，从业者、供应商和工具会要求某一些开发工作必须在隔离的环境中完成。我们反对这种观念，也在不断寻求新的方法使软件开发重回团队协作。在构建诸如软件之类的复杂工程时，反馈是至关重要的。随着项目不断提升对专业人才的需求，我们努力让他们融入常规的团队协作和反馈互动中。我们特别不提倡“10倍工程师”的理念，更喜欢专注于创建和赋能“10倍团队”。我们已经看到这在如何将设计、数据科学和安全融入跨功能团队，并获得成熟的自动化支持方面发挥了作用。下个阶段应致力于引入更多治理方法和合规行为。

THE RADAR



技术

采纳
1. Container security scanning
2. Data integrity at the origin
3. Micro frontends
4. Pipelines for infrastructure as code
5. Run cost as architecture fitness function
6. Testing using real device

试验
7. Automated machine learning (AutoML)
8. Binary attestation
9. Continuous delivery for machine learning (CD4ML)
10. Data discoverability
11. Dependency drift fitness function
12. Design systems
13. Experiment tracking tools for machine learning
14. Explainability as a first-class model selection criterion
15. Security policy as code
16. Sidecars for endpoint security
17. Zhong Tai

评估
18. BERT
19. Data mesh
20. Ethical bias testing
21. Federated learning
22. JAMstack
23. Privacy-preserving record linkage (PPRL) using Bloom filter
24. Semi-supervised learning loops

暂缓
25. 10x engineers
26. Front-end integration via artifact
27. Lambda pinball
28. Legacy migration feature parity

平台

采纳

试验
29. Apache Flink
30. Apollo Auto
31. GCP Pub/Sub
32. Mongoose OS
33. ROS

评估
34. AWS Cloud Development Kit
35. Azure DevOps
36. Azure Pipelines
37. Crowdin
38. Crux
39. Delta Lake
40. Fission
41. FoundationDB
42. GraalVM
43. Hydra
44. Kuma
45. MicroK8s
46. Oculus Quest
47. ONNX
48. Rootless containers
49. Snowflake
50. Teleport

暂缓

工具

采纳
51. Commitizen
52. ESLint
53. React Styleguidist

试验
54. Bitrise
55. Dependabot
56. Detekt
57. Figma
58. Jib
59. Loki
60. Trivy
61. Twistlock
62. Yocto Project

评估
63. Aplas
64. asdf-vm
65. AWSume
66. dbt
67. Docker Notary
68. Facets
69. Falco
70. in-toto
71. Kubeflow
72. MemGuard
73. Open Policy Agent (OPA)
74. Pumba
75. Skaffold
76. What-If Tool

暂缓
77. Azure Data Factory for orchestration

语言&框架

采纳

试验
78. Arrow
79. Flutter
80. jest-when
81. Micronaut
82. React Hooks
83. React Testing Library
84. Styled components
85. Tensorflow

评估
86. Fairseq
87. Flair
88. Gatsby.js
89. GraphQL
90. KotlinTest
91. NestJS
92. Paged.js
93. Quarkus
94. SwiftUI
95. Testcontainers

暂缓
96. Enzyme

技术

Container security scanning
采纳

持续采用容器的方式进行部署，尤其是Docker，让容器安全扫描变成了必不可少的技术，我们已将该技术移至“采纳”中以体现这一点。具体来说，容器为安全问题带来了一条新的途径，在部署过程中使用工具扫描和检查容器尤为重要。我们更愿意将自动化扫描工具的运行作为部署流水线的一部分。

Data integrity at the origin
采纳

今天，许多组织对如何解锁用于分析的数据的解决方案是建立迷宫般的数据管道。管道从一个或多个数据源检索，清理，然后转换数据并将其移动到另一个位置以供使用。这种数据管理方法通常会让使用数据的管道承担一项困难的任务，即验证入站数据的完整性，并构建复杂的逻辑来清理数据，以满足所需的质量级别。根本的问题是，数据源没有为其消费者提供高质量数据的动机和责任。出于这个原因，我们强烈主张从源头保证数据完整性，我们的意思是，任何提供可消费数据的源都，必须明确地描述其数据质量的标准，并确保这些标准。这背后的主要原因是，原始系统和团队最熟悉他们的数据，并且最适合在源头修复它。数据网格架构更进一步，将可消费数据与产品相比较，其中的数据质量及其目标是每个共享数据集的整体属性。

Micro frontends
采纳

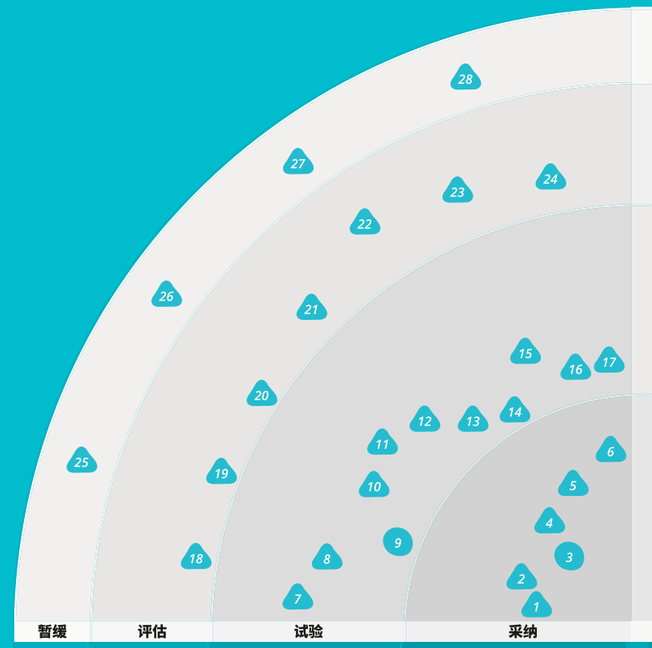
引入微服务令我们受益匪浅，使用微服务，团队可以扩展那些独立部署及维护的服务的交付。遗憾的是，我们也看到许多团队创建了单体前端——一个建立在后端服务之上的大而混乱的浏览器应用程序——这在很大程度上抵消了微服务带来的好处。自从问世以来，微前端持续变得流行。我们已经看到，许多团队采用这种架构的某种形式，来管理多开发人员和多团队的复杂性，以提供相同的用户体验。在今年的六月份，这个技术的发起人之一，发表了一篇介绍性的文章，可以起到微前端参考文献的作用。它展示了这种设计是如何通过各种Web编程机制实现的，以及使用React.js构建了一个示例应用程序。我们有理由相信，随着大型组织尝试在跨多团队中分解UI开发，这种风格将越来越流行。

采纳
1. Container security scanning
2. Data integrity at the origin
3. Micro frontends
4. Pipelines for infrastructure as code
5. Run cost as architecture fitness function
6. Testing using real device

试验
7. Automated machine learning (AutoML)
8. Binary attestation
9. Continuous delivery for machine learning (CD4ML)
10. Data discoverability
11. Dependency drift fitness function
12. Design systems
13. Experiment tracking tools for machine learning
14. Explainability as a first-class model selection criterion
15. Security policy as code
16. Sidecars for endpoint security
17. Zhong Tai

评估
18. BERT
19. Data mesh
20. Ethical bias testing
21. Federated learning
22. JAMstack
23. Privacy-preserving record linkage (PPRL) using Bloom filter
24. Semi-supervised learning loops

暂缓
25. 10x engineers
26. Front-end integration via artifact
27. Lambda pinball
28. Legacy migration feature parity



技术

自动化学习（AutoML）工具的出现填补了市场上对于相关专业技能人士的供需缺口，这类工具是一个不错的起点，但仍然需要专业人士的应用才能发挥出最好的效果。

（AutoML）

二进制鉴证就是一项实现部署时安全控制的技术，用密码学技术验证部署用的二进制镜像。

（Binary attestation）

Pipelines for infrastructure as code

采纳

使用持续交付流水线来编排软件的发布过程已经成为主流观念。CI/CD工具能够被用来测试服务器的配置（如Chef cookbooks，Puppet modules和Ansible playbooks），服务器的镜像构建（如Packer），环境的生成（如Terraform，CloudFormation）和环境间的集成。将流水线用于基础设施即代码，可以让你在变更应用于运行环境（包括开发和测试环境）之前就发现错误。它们还提供了CD/CD代理（Agent）而非独立工作站的方式，来确保基础设施工具运行的一致性。我们团队在项目中采用这种技术取得了良好的效果。

Run cost as architecture fitness function

采纳

对于今天的组织来说，自动化评估、跟踪和预测云基础设施的运行成本是必要的。云供应商精明的定价模型，以及基于定价参数的费用激增，再加上现代架构的动态本质，常常导致让人吃惊的运行成本。例如，无服务架构基于API访问量的费用，事件流方案中基于流量的费用，以及数据处理集群中基于运行任务数量的费用，它们都具有动态的本质，会随着架构演进而产生改变。当我们的团队在云平台上管理基础设施时，将运行成本实现为架构适应度函数是他们的早期活动之一。这意味着我们的团队可以观察运行服务的费用，并同交付的价值进行对比；当看到与期望或可接受的结果之间存在偏差时，他们就会探讨架构是否应该继续演进了。对运行成本的观察和计算需要被实现为自动化的函数。

Testing using real device

采纳

当成功采用持续交付后，团队尽可能让各种测试环境更接近于生产环境。这样他们就可以避免那些只会在产品环境中暴露的缺陷。这对于嵌入式和物联网软件也有效。如果我们不在真实的环境中测试，可预见的是，我们将在产品环境中首次发现某些缺陷。使用真实设备进行测试，在持续交付流水线中确保提供正确的设备，可以帮助避免这个问题。

Automated machine learning (AutoML)

试验

机器学习的强大能力和远大前途使得对专业人才的需求远远超出了专门从事该领域的数据科学家的数目。针对这种技能上的差距，我们看到了自动化机器学习（AutoML）工具的出现，这类工具旨在帮助非专业人士更容易地自动化完成从模型选择到模型训练的端到端过程。比如Google的AutoML，DataRobot和H2O AutoML Interface。尽管我们已经从这些工具中看到了可喜的成果，但还是要提醒企业不要将其视为机器学习旅程的全部。如H2O网站所述，“在数据科学领域，仍需要相当深厚的知识和经验才能产出高性能的机器学习模型”。对自动化技术的盲目信任，还会增加引入道德偏见或做出不利于少数群体的决策风险。虽然企业可以使用这些工具作为起点，生成基本有用的经过训练的模型，但我们还是鼓励他们寻找经验丰富的数据科学家来验证和完善最终的模型。

Binary attestation

试验

随着容器的广泛使用，由自治团队部署大型服务并以越来越快的速度持续交付，在许多组织已变成一种通用实践，这也导致对自动部署时软件安全控制的需求增多。二进制鉴证就是一项实现部署时安全控制的技术，用密码学技术验证部署用的二进制镜像。使用这项技术，一个签证人，一个自动构建流程，或者一个安全小组可以签发已经通过安全检查、测试，并得到授权的待部署镜像。支持在部署前创建证明和验证镜像签名的，除了Grafeas的GCP Binary Authorization服务，还有in-toto和Docker Notary这样的工具。

Continuous delivery for machine learning (CD4ML)

试验

随着基于ML的应用程序的日益普及以及构建它们所涉及的技术复杂性，我们的团队严重依赖于机器学习的持续交付（CD4ML），以安全快速且可持续的方式交付此类应用程序。CD4ML是将CD原理和实践引入ML应用程序的学科。它消除了从训练模型到部署生产环境的长周期。在构建和部署模型的端到端过程中，CD4ML消除了不同团队、数据工程师、数据科学家和ML工程师之间的手

动传递。使用CD4ML，我们的团队成功地实现了基于ML的应用程序所有组件的自动化版本管理，测试和部署，包括数据，模型和代码。

Data discoverability

试验

数据科学家和分析师在工作流程中遇到的主要问题之一是找到所需的数据，弄清楚数据含义，并评估其是否值得使用。由于缺少可用数据源的元数据，并且缺少搜索和定位数据所需的适当功能，因此这仍然是一个挑战。我们鼓励提供分析数据集或构建数据平台的团队以数据的可发现性为其生态圈的首要功能；提供轻松定位可用数据，检测其质量，了解其结构和源头，并获得访问权的能力。传统上，此功能是由庞大的数据分类解决方案提供的。近年来，我们已经看到相关开源项目的增长，这些项目正在改善数据提供者和数据消费者的开发体验，从而使他们真正做好一件事：使数据易于发现。这些工具包括Lyft的Amundsen和LinkedIn的WhereHows。我们希望看到的改变是，提供者有意识地分享利于可发现性的元数据，从而帮助发现性工具从应用程序数据库中推断出部分元数据信息。

Dependency drift fitness function

试验

许多团队和组织缺少正式或一致的方式来跟踪软件中的技术依赖关系。当软件需要更改，而其中使用的过时版本的库、API或者组件引发问题或者延期时，这个问题就暴露出来了。依赖漂移适应度函数是一项引入了特定的演进式架构适应度函数的技术，它能随着时间推移追踪这些依赖，从而能够指出可能需要的工作，以及某个潜在问题是在好转还是恶化。

Design systems

试验

随着应用程序开发变得越来越动态和复杂，高效地交付风格一致可访问和可用的产品变成了一项挑战。设计系统定义了一套设计模式、组件库以及良好的设计和工程实践的集合，以确保数字产品开发的一致性。在跨团队和学科的产品开发中，我们发现设计系统是对工具箱的有用补充，因为它们让团队可以专注在产品本身更具战略性的挑战上，而无需在每次需要添加一个视觉组件时都不得不重新发明轮子。你用于创建设计系统的组件和工具的类型都可能存在很大的不同。

Experiment tracking tools for machine learning

试验

机器学习的日常工作通常可以归结为一系列的实验，包括选择建模方法，网络拓扑，训练数据集以及对模型的各种优化或调整。由于其中许多模型仍然难以解析或解释，因此数据科学家必须使用经验和直觉来假设一些改变，然后测量这些变化对模型整体性能的影响。随着这些模型在业务系统中使用得越来越普遍，出现了几种不同的”用于机器学习的实验跟踪工具”，以帮助研究人员有条理地进行实验，并跟踪这些实验结果。尽管该领域还没有明确的赢家出现，但是诸如MLflow和Weights&Biases之类的工具，Comet和Neptune之类的平台，已经在整个机器学习工作流程中引入了严谨性和可重复性。除此之外，它们还促进了相互协作，将数据科学从一项单独的工作转变为一项团队协作的运动。

Explainability as a first-class model selection criterion

试验

深度神经网络在很多问题上都表现出了惊人的记忆力和准确性。只要有足够的训练数据和适当拓扑选择，这些模型就能满足并超越某些特定问题域中的人类能力。然而，它们天生是不透明的。虽然模型的某些部分可以通过迁移学习进行重用，但是我们很少能够赋予这些元素人类可理解的意义。相比之下，可解释的模型是一个允许我们说明决策是如何做出的模型。例如，一个决策树产生描述分类过程的推理链。可解释性在某些受监管的行业，或当我们关注决策的道德影响时变得至关重要。由于这些模型被更广泛地合并到关键的业务系统中，因此将可解释性作为模型选择的头等标准非常重要。尽管功能强大，神经网络在可解释性要求严格的情况下，也可能不是一个合适的选项。

技术

深度神经网络在很多问题上都表现出了惊人的记忆力和准确性。但是，随着使用量的增加，对如何达成决策进行阐释也越来越重要。

（Explainability as a first-class model selection criterion）

技术

当今技术环境的复杂性要求将安全策略视为代码；我们需要定义安全策略脚本、将其加入版本控制中、自动验证、自动部署并监测其性能。

(Security policy as code)

中台的核心是提供封装业务模型的方法。它旨在帮助新型的小型企业提供一流的服务，而无需传统企业基础架构的成本。

(Zhong Tai)

Security policy as code

试验

安全策略是保护我们的系统免受威胁和破坏的规则和程序。例如，访问控制策略定义并强制谁可以在什么情况下访问哪些服务和资源；或者网络安全策略可以动态地限制特定服务的流量速率。当今技术环境的复杂性要求将安全策略视为代码；我们需要定义安全策略脚本、将其加入版本控制中、自动验证、自动部署并监测其性能。Open Policy Agent这样的工具或者Istio之类的平台提供了灵活的策略定义和实施机制，它们都可以支持安全策略即代码的实践。

Sidecars for endpoint security

试验

我们今天构建的许多技术解决方案，都运行在日益复杂的多云或混合云环境中，其中包含多个分布式组件和服务。在这种情形下，我们在实施初期应用了两个安全原则：零信任网络，永远不要信任网络并始终进行验证；以及最小权限原则，即授予执行特定作业所需的最小权限。端点安全性的边车(Sidecars for endpoint security)是实现这些原则的一种常用技术，用于在每个组件的端点上实施安全控制，例如服务、数据存储和Kubernetes控制接口的API。我们使用进程外的边车来实现——一个共享相同执行上下文、主机和标识的运行中的进程或容器。开放策略代理(Open Policy Agent)和Envoy是实现此技术的工具。用于端点安全的Sidecars将可信的足迹最小化到本地端点，而不是整个网络。最后，我们希望由负责端点的团队负责sidecar安全策略的配置，而不是单独的中心化团队。

Zhong Tai

试验

近年来，中台一直是中国IT界的流行语，但它尚未在西方国家流行起来。中台的核心是提供封装业务模型的方法。它旨在帮助新型的小型企业提供一流的服务，而无需传统企业基础架构的成本，并使现有组织能够以惊人的速度将创新服务推向市场。中台战略最初是由阿里巴巴提出的，并很快被许多中国的互联网公司所采用，因为它们的商业模式是数字原生的，可以复制到新的市场和领域。如今，越来越多的中国公司将中台作为数字化转型的杠杆。

BERT

评估

BERT代表来自变换器的双向编码器表征量。它是Google在2018年十月份提出的一种新的预训练表示方法。BERT通过获得各种自然语言处理（NLP）任务的最高水平结果，极大地改变了自然语言处理（NLP）的格局。基于转换器架构，它在训练期间从令牌的左右预警中学习。Google还发布了经过预训练的通用BERT模型，该模型已针对包括Wikipedia在内的大量未标记文本进行了训练。开发人员可以在其特定于任务的数据上，使用和微调这些经过预训练的模型，并获得出色的结果。我们已经在2019年四月份的技术雷达上讨论过NLP的迁移学习；BERT以及它的后继者会继续使NLP的迁移学习成为一个令人兴奋的领域，NLP的迁移学习可以大大减少处理文本分类的用户工作量。

Data mesh

评估

数据网格是一种可以解锁大规模数据分析的架构范式；快速解锁对越来越多的分布领域数据集的访问，从而支撑跨组织的大量数据的使用场景，如机器学习、分析或数据密集型应用程序。数据网格解决了传统集中式数据湖或数据平台体系结构的常见故障模式，它改变了数据湖及其前身数据仓库的集中式范式。数据网格的范式借　鉴了现代分布式体系结构：将领域作为首要关注点，应用平台思维来创建自服务的数据基础设施，将数据视为产品，并应用开放标准化，从而实现可交互的分布式数据产品生态系统。

Ethical bias testing
评估

在过去的一年，我们已经看到人们对机器学习尤其是深度神经网络的兴趣正在发生变化。到目前为止，这些模型的卓越功能推动了工具和技术上令人兴奋的发展。虽然目前，人们越来越担心这些模型可能会造成意外伤害。例如，一个模型可以经过训练，通过简单地排除弱势申请人，而做出有利可图的信用决策。幸运的是，我们看到人们对道德偏见测试的兴趣与日俱增，这将有助于发现潜在的有害决策。一些工具，例如lime，AI Fairness 360或者What-If，可以帮助我们发现一些训练数据和可视化工具中未被充分代表的群体而导致的不准确性。可视化工具中，Google Facets和Facets Dive可以用来发现大量训练数据中的子组。但是，这是一个正在发展的领域，我们期待随着时间的推移，出现针对道德偏见测试的标准和实践。

Federated learning
评估

模型训练通常需要从数据源收集大量的数据，并将这些数据传输到集中运行模型训练算法的服务器上。但是如果训练数据集中包括个人身份信息，这就会成为问题。而联邦学习这项技术的出现让我们备受鼓舞。联邦学习是一种隐私保护方法，用于训练大量的，与个人信息相关的各种数据集。联邦学习技术可以让数据保留在用户的设备上，并完全控制在用户的手中，但最终会仍然可以组合成一个整体的训练数据集。在联邦学习中，每个用户设备独立地更新模型。然后将模型的参数（而不是数据本身）组合成一个集中式的视图。尽管网络带宽和设备算力限制会给这项技术带来一些重大的技术挑战，但是我们喜欢联邦学习的思路，让用户可以完全控制自己的个人信息。

JAMstack
评估

许多年前从手机原生开发兴起的后端即服务开发模式，现在在Web开发上变得流行起来。我们将这种集合了静态站点生成和利用第三方API进行客户端渲染的框架被称为JAMstack（JAM代表**J**avaScript，**A**PI和**M**arkup），例如Gatsby.js。这种方式之所以能给用户提供丰富的体验，主要依靠的是API和SaaS。因为HTML不管是在网页浏览器中还是在构建时渲染，它的部署模型和全静态生成的网站是一样的，共同的好处是服务端的攻击面很小，而使用很少的资源可以获得极好的性能。事实上，像这种在部署上对内容发布网络（CDN）非常友好的技术，我们开玩笑想把它称为CDN优先应用程序。

Privacy preserving record linkage（PPRL）using Bloom filter
评估

在使用共享密钥的场景下，不同数据提供者之间的记录连接是很容易实现的。但是你可能并没有一个共享密钥；即使有，基于隐私的考虑也不建议公开它。使用布隆过滤器（Bloom filter，一种节省空间的概率数据结构）建立保护隐私的记录连接（PPRL）是一种成熟的技术，它允许来自不同数据提供者进行概率记录链接，而不会公开私密的个人身份资料。例如，当连接来自两个数据提供者的数据时，每个提供者使用布隆过滤器，加密其个人身份数据以获得加密链接密钥，然后通过安全通道将它们发送给你。一旦接收到数据，就可以通过计算来自每个提供者的加密链接密钥之间的相似度得分，来链接这些记录。与其他技术相比，我们发现使用布隆过滤器的PPRL对于大型数据集是可伸缩的。

Semi-supervised learning loops
评估

半监督学习循环是一类迭代式的机器学习工作流，它们利用未标记数据中尚待发现的关系，来提升学习性能。这些技术通过不同方式组合标记和未标记的数据集，从而改进模型。此外，它们还对在不同数据子集上训练出来的的模型进行对比。与机器从未标记数据中推断分类的无监督学习，以及训练集完全标记的有监督技术不同，半监督技术利用的是一小部分被标记数据和大部分未标记数据。半监督学习还与主动学习技术密切相关，在主动学习技术中，人们被引导至选择性标记的模糊数据点。因为能够精确标记数据的专家是稀缺资源，并且标记通常是机器学习中最耗时的活动，所以半监督技术不仅降低了训练成本，还使机器学习对于新型用户而言是可行的。

技术

联邦学习这项技术的出现让我们备受鼓舞。联邦学习是一种隐私保护方法，用于训练大量的，与个人信息相关的各种数据集。

（Federated learning）

JAMstack之所以能给用户提供丰富的体验，主要依靠的是API和SaaS。

（JAMstack）

技术

根据我们的经验，伟大的工程师不是因为个人产出而是因为能在优秀的团队中合作而诞生。

（10x engineers）

10x engineers

暂缓

在过去的几个月中，10倍工程师一词受到了密切的关注。一个广泛传播的推文讨论在实质上建议公司应原谅反社会和破坏性的行为，以留住被认为个人产出巨大的工程师。幸运的是，许多人在社交媒体上都嘲笑了这个概念，但是“明星开发者”的刻板印象仍然普遍存在。根据我们的经验，伟大的工程师不是因为个人产出而是因为能在优秀的团队中合作而诞生。打造一支混合不同经验和背景，但成员才华横溢的团队，并为团队合作、学习和持续改进提供良好的助力，这会是更行之有效的方式。这些10倍团队行动起来更快，弹性也更强——而无需屈从错误的行为。

Front-end integration via artifact

暂缓

当团队接受微前端这个概念时，他们有很多种方式将各个微前端集成到一个应用程序中，同样也会有一些反模式。其中最常见的一种方式就是通过制品进行前端集成。每一个微前端会被构建成一个制品，这个制品通常是一个被推送到注册表中的NPM软件包。接下来，在不同的构建流水线中，将各个包组合成一个最终的软件包，这个软件包包含了所有的微前端。从纯粹的技术角度来看，这种集成方式可以使应用程序正常运行。但是，通过制品的方式进行集成，意味着每一次修改都需要重建整个包。这不仅耗时，还有很大可能会带来负面的开发体验。更糟糕的是，这种集成方式会引入构建过程中微前端的直接依赖关系，从而导致相当大的协调开销。

Lambda pinball

暂缓

在项目中构建无服务器架构的这几年，我们注意到落入一个分布式单体应用的陷阱是一件很容易的事情。当请求在日益复杂的云服务中不断跳转时，深陷Lambda、存储桶与队列中的Lambda弹球架构明显无法看清重要的领域逻辑，通常只能对应用整体进行集成测试，而很难进行单元测试。为了避免陷入弹球架构，我们可以明确区分公共与发布的接口，沿用良好的领域边界，并在其间使用已发布的接口。

Legacy migration feature parity

暂缓

我们发现，越来越多的组织需要替换陈旧的遗留系统，以适应其客户（内部和外部）的需求。我们持续看到的一种反模式是遗留系统迁移的功能一致性，即保留旧版本同样功能的愿望。我们认为这错失了一个巨大的机会。随着时间流逝，旧系统往往会变得臃肿，包含了许多不被用户使用的功能（根据2014年Standish Group的报告，这一比例为50%）和随着时间而发展的业务流程。替换这些功能是一种浪费。我们的建议：说服你的客户退一步思考，了解他们的用户当前需要什么，并根据业务结果和指标，对这些需求进行优先排序——这通常说起来容易做起来难。这意味着需要进行用户调研，并采用现代产品开发实践，而不是简单地替换现有的系统。

平台

Apache Flink
试验

自从Apache Flink在2016年首次进入技术雷达“评估”环以来，越来越多的人开始采用它了。Flink被视为领先的流处理引擎，在批处理与机器学习领域也逐渐成熟。与其他流处理引擎相比，Flink的独特之处在于，它使用了一致的应用状态检查点。当发生错误时，应用可以重启，并从最近的检查点载入状态继续处理，就好像错误从未发生一样，这让我们不必为了容错而不得不构建和操作复杂的外部系统。我们看到越来越多的公司，在使用Flink构建他们的数据处理平台。

Apollo Auto
试验

Apollo Auto让自动驾驶技术不再是科技巨头独享的”火箭科技”。百度旗下的Apollo项目的目标，是成为自动驾驶产业里的安卓操作系统。Apollo平台拥有诸如感知、模拟、规划以及智能控制相关的一系列组件，能够让汽车公司将他们的自动驾驶系统集成到汽车硬件中。虽然开发者社区刚刚起步，但是许多第三方厂商陆续加入并贡献良多。我们的一个项目就是帮助客户，使用基于Apollo的自动驾驶系统，完成自动驾驶执照的考试。Apollo同时也提供演进式架构的方法，以逐步引入先进功能，让我们能以敏捷和迭代的方式，集成更多的传感器和功能。

GCP Pub/Sub
试验

GCP Pub/Sub是谷歌云的事件流平台。对于谷歌云平台上我们所使用的许多架构而言，由于具备大规模事件提取、无服务器工作负载通信以及具备能进行流数据处理的工作流等特性，这一基础设施广受欢迎。该平台另一个独特功能，是支持拉和推两种事件订阅方式，即在订阅期间能接收所有可用的已发布消息，并能将消息推送到特定端点。我们很喜欢该平台的可靠性和伸缩性，其工作表现名实相符。

Mongoose OS
试验

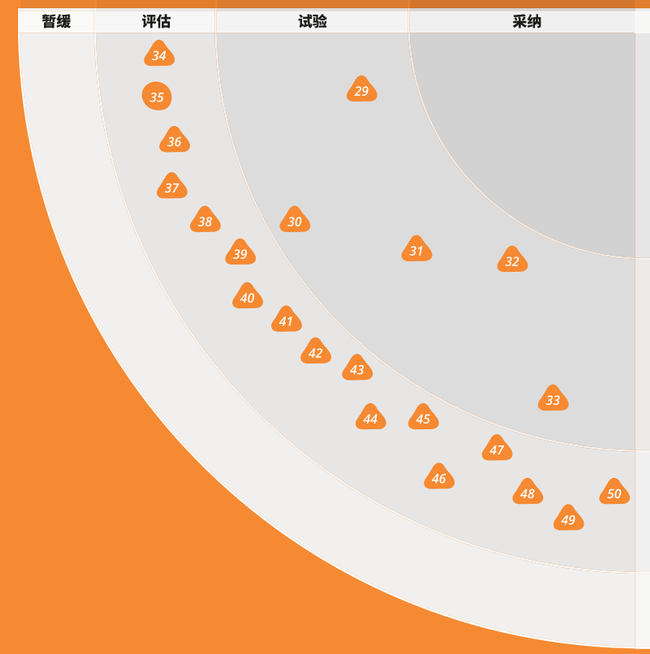
Mongoose OS依然是我们首选的开源微控制器操作系统和嵌入式固件开发框架。值得关注的是，Mongoose OS为嵌入式开发人员解决了一个明显的问题，即消除了用于原型设计的Arduino固件与裸机微控制器原生SDK之间所存在的冲突。我们的团队已成功将Cesanta的新型端到端设备管理平台mDash，用于一些新的小型硬件项目中。如今，主要的物联网（IoT）云平台供应商，都支持Mongoose OS开发框架，以进行设备管理、连接控制和空中（OTA）固件升级。自从我们上次将Mongoose OS纳入技术雷达以来，意法半导体、德州仪器和乐鑫科技等公司所生产的板卡和微控制

采纳

试验
29. Apache Flink
30. Apollo Auto
31. GCP Pub/Sub
32. Mongoose OS
33. ROS

评估
34. AWS Cloud Development Kit
35. Azure DevOps
36. Azure Pipelines
37. Crowdin
38. Crux
39. Delta Lake
40. Fission
41. FoundationDB
42. GraalVM
43. Hydra
44. Kuma
45. MicroK8s
46. Oculus Quest
47. ONNX
48. Rootless containers
49. Snowflake
50. Teleport

暂缓



平台

Apache Flink是领先的流处理引擎，并且在批处理和机器学习领域也日趋成熟。

（Apache Flink）

百度旗下的Apollo项目的目标，是成为自动驾驶产业里的安卓操作系统。

（Apollo Auto）

器，都开始对其提供了支持。其对OTA固件更新的无缝支持，以及单设备级别的内置安全性，都将继续受到我们的青睐。

ROS

试验

ROS（Robot Operating System，机器人操作系统）这套程序库和工具集，能帮助软件开发人员创建机器人应用程序。它也是一个开发框架，提供硬件抽象、设备驱动、程序库、可视化程序、消息传递、包管理等功能。开放自动驾驶平台Apollo Auto正是基于ROS的。在另一个ADAS（高级驾驶辅助系统）模拟项目中，我们还使用了ROS的消息传递系统（使用bag文件格式）。随着ADAS的发展，ROS这项不算新的技术，又重新引发开发人员的关注。

AWS Cloud Development Kit

评估

尽管我们很多团队，已经将Terraform作为定义云基础设施的默认选择，但另一些团队却已经开始尝试使用AWS Cloud Development Kit（AWS CDK），并对其青睐有加。他们尤其喜欢该工具使用编程语言而不是配置文件来定义云基础设施，这样就能使用已有的工具、测试方法和技能来进行工作。即使对于定义云基础设施这样的工具，也需要用心确保部署代码易于理解和维护。考虑到该工具很快就会支持C#和Java语言，并且暂时忽略某些功能上的差距，如果希望不使用配置文件来定义云基础设施，我们认为AWS CDK值得关注。

Azure DevOps

评估

Azure DevOps服务包括一组被托管的服务，如Git仓库、CI/CD流水线、自动化测试工具、待办列表管理工具和制品仓库。Azure DevOps Pipelines已经日趋成熟，其对流水线即代码的支持，以及在Azure DevOps市场中的生态系统的扩展能力，尤其赢得我们的青睐。但在撰写本文时，我们的团队还是发现它的一些功能不够成熟，比如缺少有效的UI，来进行流水线的可视化和导航，以及无法从制品或其他流水线，来触发某条流水线。

Azure Pipelines

评估

Azure Pipelines是Azure DevOps套件中的一款产品，提供了基于云的解决方案，能为Azure DevOps Git或其他Git解决方案（如Github或Bitbucket）中的项目，实现流水线即代码的功能。该解决方案的亮点在于，能将脚本运行在Linux、MacOS和Windows代理（agent）上，而无须自行管理虚拟机。这对于团队（尤其是在Windowks环境下使用.NET框架的团队）来说，意味着前进了一大步。我们也正在评估此服务在iOS方面的持续交付能力。

Crowdin

评估

开发团队在开发大多数具有多语言支持的项目时，往往一开始只构建一种语言的功能，然后以电子邮件和电子表格形式，离线翻译其余语言。这种简单的安排虽然可行，但很快就会失控。因为可能需要为不同的语言翻译者，重复回答相同的问题，从而使译者、校对人员和开发团队，耗费大量精力进行协作。Crowdin是少数几个同类平台中的一员，能简化项目本地化工作流程的。开发团队在持续构建功能的同时，Crowdin平台可以持续将待翻译文本，整合至在线工作流中。我们喜欢Crowdin能推动团队持续和渐进地整合翻译工作，从而避免在最后阶段才进行大批量的处理。

Crux

评估

Crux是一个带有双时态图查询的开源文档数据库。许多数据库系统都是有时态的，这意味着它们可以帮助我们对事实及其所发生的时间进行建模。双时态数据库系统不仅可以对事实所发生的有效时间进行建模，还可以对收到事实并产生交易的时间进行建模。如果需要使用具有图查询功能的文档库来查询内容，可以试试Crux。虽然该数据库当前还处在alpha测试阶段，且缺乏SQL支持，但是可以使用Datalog查询界面来读取和遍历关系。

Delta Lake
评估

Delta Lake是一个由Databricks开发的开源存储层，用于在大数据场景中引入事务处理。我们在使用Apache Spark时经常遇到的一个问题是缺少ACID事务。Delta Lake通过与Spark的API集成，使用事务日志和版本化的Parquet文件解决了这个问题。由于其可序列化的隔离性，它允许读取器和写入器对Parquet文件进行并发的操作。它的另一个广受好评的特性是对写操作和版本控制的模式强制，它允许我们在必要时可以查询和恢复到旧版本的数据。Delta Lake已经在我们的一些项目中得到了应用，并收获了极好的用户评价。

Fission
评估

Kubernetes的无服务器生态系统正在增长。我们在之前的雷达中讨论过Knative，现在我们看到Fission获得了关注。Fission让开发人员可以专注于编写短期的函数并将它们映射到HTTP请求，而Kubernetes资源的自动化和其他工作则交给框架幕后处理。Fission还允许组合功能，通过webhooks与第三方提供商集成，并自动管理Kubernetes基础设施。

FoundationDB
评估

FoundationDB是一个开源的多重模型数据库。在被苹果公司收购三年后，于2018年4月进行了开源。它的核心是一个提供严格可序列化操作的分布式的键值存储。比较有意思的地方是FoundationDB是它的扩展层的概念。扩展层是在其核心键值存储之上的一些无状态组件，例如记录层和文档层。同时FoundationDB为模拟测试设置了很高的标准，它们每天运行这些测试来模拟各种系统故障。出色的性能、严谨的测试和简单的操作使得它不单只是一个数据库，有人也用它来作为核心底层，并基于此开发分布式系统。

GraalVM
评估

GraalVM是一种由Oracle开发的通用虚拟机，用于运行基于JVM的语言，JavaScript，Python，Ruby和R以及C/C++等其他基于LLVM的语言编写的应用程序。简单地说，GraalVM可以用作VM和其他所支持的非JVM的语言的高性能虚拟机。但它也允许我们编写多种语言的应用程序，而且对性能影响很小;它的原生镜像程序（当前仅可作为早期采用者的技术使用）让我们可以提前将Java代码编译成独立的可执行文件，从而加快启动速度并减少内存的使用。GraalVM在Java社区引起了巨大的反响，并且许多Java框架（包括Micronaut，Quarkus和Helidon）已经在利用它的技术了。

Hydra
评估

并非每个人都需要一个自托管的OAuth2解决方案，但是如果你需要，我们发现Hydra是一个非常有用的方案，它是一个完全兼容开源OAuth2认证服务器和OpenID Connect的服务提供方。我们非常喜欢它，因为它不提供开箱即用的认证管理解决方案，无论采用哪种身份管理方式，都可以通过干净的API与其集成。身份认证管理与OAuth2框架其余部分之间的清晰区分，使Hydra与现有身份认证生态系统的集成更加容易。

Kuma
评估

Kuma是可用于Kubernetes、VMs和裸机环境的，与平台无关的服务网格。它基于Envoy实现了一个控制平面来监测任何4层/7层的网络流量，以此保护、观察、路由和增强服务之间的连接性。大多数服务网格的实现都是针对Kubernetes生态系统的，这本身没问题，但是会阻碍现有的非Kubernetes应用程序使用服务网格。现在，你可以通过Kuma来使网络基础设施现代化，而不是花费精力完成大量的平台迁移工作。

MicroK8s
评估

过去我们讨论过Kubernetes，它仍是在生产集群中部署和管理容器的默认选择。但为开发人员提供类似的本地体验变得越来越困难。在一些其他选项中，我们发现MicroK8s非常有用。安装MicroK8s snap，需要选择一个发布渠道（稳定版、候选版、Beta版或Edge版），使用一些命令来让Kubernetes运行起来即可。你也可以持续关注主流版本的发布，并选择是否自动化升级。

平台

GraalVM在Java社区引起了巨大的反响，并且许多Java框架（包括Micronaut，Quarkus和Helidon）已经在利用它的技术了。

（GraalVM）

Kuma是可用于Kubernetes、VMs和裸机环境的，与平台无关的服务网格。

（Kuma）

平台

在神经网络生态系统中，工具和框架之间的彼此协作性一直是一个难题。ONNX可以提供帮助。

（ONNX）

Teleport是用于远程访问云原生基础架构的安全网关。

（Teleport）

Oculus Quest

评估

我们在雷达中对AR/VR（增强现实/虚拟现实）技术关注了很长时间，但其仅仅在特定的平台及外接方案上展现了吸引力。Oculus Quest打破了这个局面，成为首批面向大众消费市场的无需智能手机绑定或支持的VR一体机之一。该设备为潜在的VR应用的大量增长打开了大门，其需求将反过来推动市场朝着更积极创新的方向发展。我们很欣赏该设备为VR普及带来的推动力，迫不及待地想看到即将到来的变化。

ONNX

评估

目前，围绕神经网络的相关工具和框架的生态系统正在迅速地发展。但是，这些框架和工具之间的互通性也成为一个挑战。在机器学习领域，通常需要在一种工具中快速进行原型设计和训练，然后将其部署到其他工具中进行推理。因为这些工具的内部格式并不兼容，为了使他们兼容，我们需要实现并维护很多麻烦的转换器。开放神经网络交换格式ONNX的出现，就是为解决这一问题。在ONNX中，表示神经网络的图形由标准规格的操作符和一系列表示训练权重和神经网络模型的格式所组成，这些图形可以在不同的工具间传递。这种一致的格式带来了很多的可能性，其中之一就是Model Zoo，它是一系列基于ONNX格式的预训练模型的集合。

Rootless containers

评估

理想情况下，容器应该由各自的容器运行时管理和运行，而不应具有root权限。这不是小事，当它实现时，能够减少攻击面并避免所有类型的安全问题，特别是容器外的权限升级。无根容器在社区被讨论了很长时间，它是开放容器运行时规范及其标准实现runc的一部分，而runc是Kubernetes的基础。现在，Docker 19.03将无根容器作为一个实验性特性引入。尽管功能齐全，该特性还不能与其他部分特性兼容，比如cgroups资源控制和AppArmor安全配置文件。

Snowflake

评估

我们经常将数据仓库和中心化的基础设施联系起来，而随着围绕数据的需求不断增长，中心化的基础设施是难以扩展和管理的。然而，Snowflake是一种全新的SQL数据仓库即服务解决方案，专为云平台而构建。Snowflake拥有大量精心设计的功能，例如数据库级的原子性，结构化和半结构化数据支持，数据库内分析功能，尤其是将存储、计算和服务层明确分离，从而解决了大多数数据仓库所面临的挑战。

Teleport

评估

Teleport是用于远程访问云原生基础架构的安全网关。其吸引人的功能，是除了用作网关以外，还可以兼作基础设施的证书颁发机构（Certificate Authority，CA）。它可以颁发短期证书，并为Kubernetes基础设施（或仅为SSH）构建更丰富的基于角色的访问控制（role-based access control，RBAC）。随着人们越来越关注基础设施的安全性，更改的跟踪变得非常重要。但是，并非所有事件都需要相同级别的审计。使用Teleport，就可以对于大多数事件仅做记录，而对于拥有更多权限的root会话，则可进一步记录用户的屏幕操作。

工具

Commitizen

采纳

Commitizen是一款提升GIT提交过程效率的小工具。它会提示你提供任何必要字段，还会恰当地格式化提交信息。Commitizen内置了对多种提交规范的支持，同时还允许你定制自己的提交规则。这个简单的工具能够节约时间，而不必等待提交钩子运行检查再驳回提交。

ESLint

采纳

我们在许多项目中都使用ESLint作为标准。作为JavaScript的代码检查工具，它提供了很多规则集、推荐规则和插件，可以扩展为不同的框架或JavaScript风格。我们发现，通过在开发时对代码进行实时分析，Eslint可以极大地帮助团队在开发过程中建立和遵循代码规范。Eslint不仅可以通过实施最佳实践和代码规范，对编码实践进行标准化，还能识别代码中的漏洞。这是因为ESLint与大多数IDE都能很好地集成，并可以在编码过程中提供实时反馈。特别是它的样式规则可以自动修复代码错误，持续有效并且不会产生额外的开发成本。ESlint社区的文档很好地解释了它的编码模式，可以帮助开发人员快速掌握规则。随着ESLint变得越来越通用和强大，它已经被行业所认可。例如TypeScript团队就选择支持ESLint并与其合作，而非TSLint。

React Styleguidist

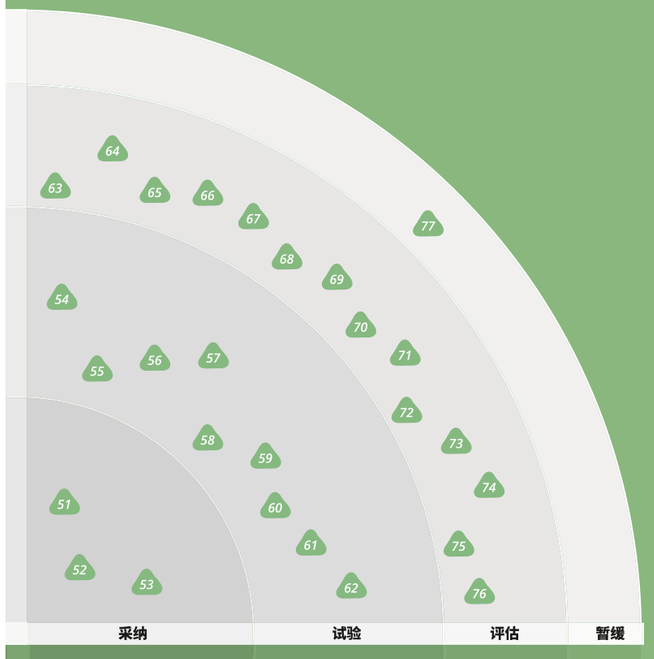
采纳

React Styleguidist是React组件的开发环境。它提供带有热重载功能的开发服务器，并可以生成HTML样式指南以便与团队进行共享。这个样式指南可以展示所有组件的最新版本，以及组件的使用文档和参数列表。我们之前在UI开发环境中介绍过React Styleguidist。而现在，相比与其他类似工具，React Styleguidist已成为我们的默认选择。

Bitrise

试验

移动应用的构建、测试和部署，尤其是将流水线从代码仓库打通到应用商店的时候，会涉及许多复杂的步骤。虽然这些步骤可以由脚本或普通CI/CD工具提供的流水线自动完成，但我们团队发现，Bitrise这个专注移动应用的持续交付工具对于不需要集成后端流水线的团队来说会更有用。它配置简单，还预置了丰富的部署步骤，可以满足绝大多数移动应用开发所需。



采纳

51. Commitizen
52. ESLint
53. React Styleguidist

试验

54. Bitrise
55. Dependabot
56. Detekt
57. Figma
58. Jib
59. Loki
60. Trivy
61. Twistlock
62. Yocto Project

评估

63. Aplas
64. asdf-vm
65. AWSume
66. dbt
67. Docker Notary
68. Facets
69. Falco
70. in-toto
71. Kubeflow
72. MemGuard
73. Open Policy Agent (OPA)
74. Pumba
75. Skaffold
76. What-If tool

暂缓

77. Azure Data Factory for orchestration

工具

Figma不仅具有与Sketch和Invision等设计程序相同的功能，而且还支持与其他人实时协作。

（Figma）

如果你要构建Java应用程序并使用Docker，则可以考虑使用Google的Jib。

（Jib）

Dependabot

试验

使代码库的依赖保持最新是一件很麻烦的事，但是出于安全考虑，及时响应依赖的更新还是很重要的。你可以使用工具让这个过程尽可能轻松和自动化。我们的团队在实际使用Dependabot时觉得不错。它可以与GitHub仓库集成，自动检查依赖的版本更新，并在必要时提交一个升级依赖的PR。

Detekt

试验

Detekt是用于Kotlin的静态代码分析工具。Dekekt可以基于高度可配置的规则集提供代码坏味道以及复杂度的检查报告。可以通过命令行运行，也可以通过Gradle、SonarQube或IntelliJ以插件方式运行。我们的团队发现了使用Detekt来维护高质量代码的巨大价值。将分析及生成报告集成到构建流水线上后，显而易见，重要的是定期检查报告，并给团队留出时间解决所发现的问题。

Figma

试验

交互和视觉设计的一大痛点是缺乏用于协作的工具，Figma就是为此而生的。它不仅具有与Sketch和Invision等设计程序相同的功能，而且还支持与其他人实时协作，帮助多人一起探索新的想法。我们的团队发现Figma非常有用，特别是它支持与简化了远程和分布式的设计工作。除了协作功能之外，Figma还提供了有助于改善DesignOps流程的API。

Jib

试验

构建容器化应用程序可能需要在开发环境和构建代理上进行复杂的配置。如果你要构建Java应用程序并使用Docker，则可以考虑使用Google的Jib。Jib是同时支持Maven和Gradle的开源插件。Jib插件使用构建配置中的信息，将应用程序直接构建为Docker镜像，而不需要Dockerfile或Docker守护程序。Jib也针对镜像分层进行了优化，可以提升后续构建的速度。

Loki

试验

Loki是一个配合Storybook使用的可视化回归工具，我们在UI开发环境中提到过Storybook。只需几行配置，就可以使用Loki测试所有UI组件。推荐在Docker容器中使用Chrome，以避免在不同的环境中运行测试时出现的1像素差异问题。我们的经验是测试非常稳定，但是Storybook的更新往往会由于细微的差异而导致测试失败。Loki似乎也无法测试使用position:fixed的组件，但可以使用fixed包装组件来规避这个问题。

Trivy

试验

我们应该在生成和部署容器的构建流水线中引入容器安全扫描。我们团队特别喜欢Trivy——一款用于容器的漏洞扫描器。它提供独立的二进制文件，相比于其他工具更容易安装和配置。而且Trivy是开源软件，并支持Distroless容器。

Twistlock

试验

Twistlock是提供构建时和运行时安全漏洞检测和预防功能的商业产品，可以保护VM、容器调度程序和容器，以及应用程序依赖的各类注册中心和存储库。Twistlock帮助我们的团队加快了受监管应用程序的开发，这些应用程序的基础设施和架构需要遵循一定的规范，例如支付卡行业（PCI）标准和《健康保险可移植性和责任法案》（HIPAA）。我们的团队很享受Twistlock带来的开发人员体验：能够以代码形式管理资源，可以轻松地与其他常见可观察性平台进行集成，以及根据行业共识最佳实践来衡量基础架构的，直接可用的基准测试。我们在例行的运行时扫描过程中，尤其是在有合规性要求的情况下，使用Twistlock对云原生应用程序进行扫描。

Yocto Project

试验

我们可以看到，越来越多强大的IOT设备选择运行Linux而不是运行一个定制的嵌入式系统。然而为了节省资源并减少攻击面，仍然有必要创建一个自定义的Linux版本，其中只包含运行设备程序的相关工具和依赖。在这种情况下可以考虑使用Yocto。它可以针对具体的需求裁剪并创建自定义的Linux发行版。Yocto的学习曲线陡峭，并且由于它的灵活性很高，很容易出错。但是经过多年发展，Yocto已经建立起了一个活跃的社区，可以起到一些帮助。与同类工具相比，它更容易被集成至持续交付的工作流。而且与Android Things和Ubuntu core不同，Yocto并没有绑定在一个指定的生态系统上。

Aplas

评估

我们的软件大厦变得越来越复杂，也越来越难以理解。Aplas是一个新的软件映射工具，可以用地图的形式可视化软件布局。Aplas首先会获取当前系统的元数据，然后在地图上投影各种视图。可以手动或通过API自动获取元数据。我们很高兴看到该产品不断发展，也看到自动收集元数据所带来的种种可能性。例如，通过公开运行成本等架构适应度函数，应该可以可视化呈现云基础设施的成本。我们经常面临的另一个问题是想要了解哪些系统用何种技术与其他系统进行通讯，这一点也可以使用Aplas帮助我们进行可视化。

asdf-vm

评估

asdf-vm是一个按项目管理多语言运行时版本的命令行工具。它与Ruby的rvm和Node的nvm等其他命令行版本管理工具类似，但更可以通过可扩展的插件体系架构支持多语言。当前插件列表包括多种语言，以及Bazel或tflint等可能需要针对每个项目管理其运行时版本的工具。

AWSume

评估

AWSume是一个方便的脚本，用于在命令行中管理AWS会话令牌及担任角色凭证。当需要同时使用多个AWS账号时，AWSume就很有用武之地。这个脚本可以从CLI缓存中读取配置，省去了在每条命令中分别指定配置的麻烦。AWSume还可以将配置输出至环境变量，这样命令与AWS SDK都能够获取到正确的凭证。

dbt

评估

数据转换是数据处理工作流的重要组成部分：筛选、分组或组合多个数据源，将它们转换为适合分析数据或机器学习模型使用的格式。dbt既是一个开源工具，也是一个商业化的SaaS产品，为数据分析师提供了简单高效的转换功能。现有的数据转换框架和工具，要么过分专注于功能强大和灵活性，却也要求对编程模型及语言框架有深刻的理解，例如Apache Spark；要么就只提供一些死板的界面拖放工具，而无法使用可靠的工程实践，如自动化测试和部署。dbt填补了这个空白：它使用被广泛理解的接口，SQL，对简单的批处理转换进行建模。同时dbt也提供了命令行工具以支持版本控制、自动化测试和部署等良好的工程实践。实际上，dbt基于SQL实现了转换模型即代码。目前，dbt支持包括Snowflake和Postgres在内的多种数据源，并提供Airflow及Apache自己的云服务等多种运行方式。dbt的转换能力受限于SQL，在撰写本文时还不支持实时的流式转换。

Docker Notary

评估

Docker Notary是对镜像、文件及容器等资产进行签名的开源工具，用于验证资产的来源。对于受控的环境来说这是超级有用的功能，而对于其他环境来说也是很好的实践。容器在创建时，会使用代表发布者身份的私钥及哈希进行签名，并存储至元数据。对于已经发布的容器（或其他资产），就可以用镜像的哈希以及发布者的公钥对其来源进行验证。虽然已经有Docker Trusted Registry这样可公开访问、可信任的注册中心，但也可以运行自己的注册中心。我们的团队在本地运行Notary服务时发现还有些问题，并建议使用其他支持Notary的注册中心。

工具

Yocto可以针对具体的需求裁剪并创建自定义的Linux发行版，例如在物联网相关的需求中。

（Yocto Project）

Aplas是一个新的软件映射工具，可以用地图的形式可视化软件布局。

（Aplas）

工具

随着人们普遍使用Kubernetes作为容器编排器，与容器和Kubernetes相关的安全工具也在快速发展。Falco就是一个专注运行时安全的容器原生工具。

（Falco）

Facets

评估

不论是直接使用还是作为机器学习模型的训练输入，越来越多的重要决策源自于大数据集。因此了解数据中的差距、缺陷和潜在偏见十分重要。Google的Facets项目在此领域提供了两个有力工具：Facets Overview和Facets Dive。Facets Overview对数据集中的特征分布进行可视化，可以展现训练和验证集的偏斜，并且可以用于比较多个数据集；Facets Dive用于在大数据集中挖掘和可视化单个数据点，并使用不同的可视维度来探究属性之间的关系。它们都是进行道德偏差测试的有力工具。

Falco

评估

随着人们普遍使用Kubernetes作为容器编排器，与容器和Kubernetes相关的安全工具也在快速发展。Falco就是一个专注运行时安全的容器原生工具。它利用Sysdig的Linux内核监控和系统调用性能数据，可以深度洞察系统的行为，进而帮助我们发现应用程序、容器、主机或者Kubernetes编排器自身的异常行为。我们喜欢Flaco，因为它不需要植入第三方的代码或者附加其他容器就能工作。

in-toto

评估

我们注意到，越来越多的地方，尤其是在受监管的行业，为了确保软件的供应链安全会使用二进制验证。当前主流的做法，或是构建一个定制的二进制验证系统，亦或是依赖于某个云厂商提供的服务。我们高兴地看到出现了开源的in-toto项目。In-toto是一个框架，能够以密码学的方式验证软件制品生产路径上的每个组件和步骤。该项目可以与众多广泛使用的构建工具、容器审计工具和部署工具进行集成。由于软件供应链工具是一个组织的安全设施中至关重要的部分，因此我们非常喜欢in-toto。作为一个开源项目，它的行为是透明的，并且其自身的完整性和供应链也可以由社区进行验证。至于它是否会赢得足够多的用户和贡献者以在这个领域竞争，我们拭目以待。

Kubeflow

评估

Kubeflow之所以令人感兴趣，是因为两个原因。首先，它是Kubernetes Operators的创新应用，我们在2019年4月版的技术雷达中对Kubernetes Operators进行了重点介绍。其次，它提供了一种对机器学习工作流进行编码和版本控制的方法，使它可以更容易地从一个执行环境移植到另一个环境。Kubeflow由几个组件组成，包括Jupyter notebooks、数据流水线和控制工具。其中一些组件被打包成为Kubernetes Operators，以利用Kubernetes响应Pod产生的事件的能力，而Pod可以实现工作流的各个阶段。通过将程序和数据打包为容器，可以将整个工作流程从一个环境移植到另一个环境。这有利于将有用但是计算能力要求极高的工作流，从云上转移到自定义超级计算机或者张量处理器（TPU）集群中。

MemGuard

评估

如果应用程序将敏感信息（例如加密密钥）以纯文本的形式存储在内存中，很可能会被人利用成为攻击媒介而导致信息泄露。大多数基于云的解决方案通常使用HSM（hardware security modules，硬件安全模块）以避免受到此类攻击。但如果是自托管服务且无法使用HSM时，仍然希望避免这类攻击，MemGuard就非常有用了。MemGuard可以作为软件安全区，在内存中存储敏感信息。尽管MemGuard不能替代HSM，但它也实现了许多安全策略，如防止冷启动攻击，避免垃圾收集的干扰，并使用防护分页技术加固以减少泄露敏感数据的可能性。

Open Policy Agent（OPA）

评估

横跨多个技术领域统一定义和实施安全策略很有挑战。即使对于简单的应用程序，也必须使用其组件内置的安全策略配置和实施机制，来控制对容器编排器、保存服务状态的服务及数据存储等组件的访问。

我们对开放策略代理（OPA）这个尝试解决此问题的开源技术感到非常兴奋。OPA可以使用Rego策略定义语言，以代码的方式进行细粒度的访问控制与灵活的策略定义。Rego在应用程序代码之外，以分布式且不干扰用户的方式实施策略。在撰写本文时，OPA以统一而灵活的策略定义及执行实现，确保了通过Envoy边车和Kafka访问Kubernetes API和微服务API的安全性。它也可以用作任何服务的边车，用以验证访问策略或过滤响应数据。OPA背后的公司Styra针对分布式策略的集中可见性提供提供商业化的解决方案。我们希望OPA可以通过CNCF孵化计划成熟起来，并继续为多样化的数据存储等更具挑战性的策略执行场景提供支持。

Pumba

评估

Pumba是Docker的混沌测试和网络仿真工具。Pumba可以终止、停止、删除或暂停docker容器，还可以仿真网络并模拟不同的网络故障，例如延迟、数据包丢失和带宽速率限制。Pumba使用tc工具进行网络仿真，因此在容器中需要安装tc，或者需要在带有tc的边车容器中运行Pumba。在对运行在多个容器中的分布式系统做混沌测试时，无论是从本地还是在构建流水线中都可以使用Pumba。

Skaffold

评估

Google为我们带来了Skaffold——用于自动化本地开发工作流程的开源工具，同时也支持部署至Kubernetes。Skaffold会检测源代码的变更，触发工作流以构建，标记和部署到K8s集群中，并捕获应用程序日志返回命令行。工作流程支持将不同的构建和部署工具以插件化形式引入，并且还提供默认配置，使入门变得更容易。

What-If Tool

评估

机器学习世界的研究重点稍微从探索“模型能够理解什么”，转向了研究“模型是如何理解的”。由于人们担心引入偏差，或者过度泛化模型的适用性，因此开发出了What-If Tool（WIT）这样的工具。这个工具可帮助数据科学家深入研究模型的行为，并将各种功能和数据集对输出的影响进行可视化。WIT由Google引入，简化了比较模型、切片数据集、可视化构面和编辑单个数据点等任务，并可以在Tensorboard或Jupyter notebooks中使用。尽管WIT可以帮助执行分析，但研究者仍然需要对模型背后的数学和理论有深刻的理解。WIT只是数据科学家用来深入了解模型行为的工具，对于使用不当或缺乏训练不佳的算法，初级用户不应奢望有任何工具可以消除或减轻其风险或造成的损害。

Azure Data Factory for orchestration

暂缓

Azure数据工厂（ADF）是Azure目前默认的用于编排数据处理流水线的产品，可以用于数据提取，在使用不同存储类型的自有产品或者Azure服务之间复制数据，以及执行数据转换逻辑。尽管我们在使用ADF做简单的自有服务数据上云的迁移时感觉还可以，但我们不推荐使用Azure数据工厂来编排复杂的数据处理流水线。我们使用ADF的体验一直很不好，原因包括：可以通过代码实现的功能有限，因为ADF似乎优先考虑的是实现低代码开发平台的功能；可调试性和错误报告差；整个流水线的可观察性有限，因为ADF的日志记录功能无法与其他产品（如Azure Data Lake Storage或Databricks）集成，因此开发者难以实现流水线的端到端可观察性；以及只有部分地区能够使用数据源触发机制。因此，目前我们推荐使用其他开源编排工具（例如Airflow）来处理复杂的数据流水线，而仅使用ADF进行数据复制或数据快照。我们期望ADF未来会解决这些问题，以支持更复杂的数据处理工作流，并可以通过代码使用新功能。

工具

What-If Tool可帮助数据科学家深入研究模型的行为，并将各种功能和数据集对输出的影响进行可视化。

（What-If Tool）

语言&框架

Arrow
试验

Arrow是适用于Kotlin的函数式编程库，是由两个现有流行库（kategory和funKTionale）合并而成。虽然Kotlin为函数式编程提供了构建模块，但Arrow为应用程序开发人员准备了随时可用的高级抽象包。它提供数据类型、类型类、作用（Effects）、Optics和其他函数式编程模式，并且可以与流行库相集成。我们对于Arrow最初的好印象如今已经在生产环境的应用构建中得到了印证。

Flutter
试验

我们的一些团队使用了Flutter并且很喜爱它。作为跨平台框架，它可以帮助我们用Dart语言编写原生移动应用。借助Dart，Flutter可以编译成平台原生代码并直接和目标平台通讯，从而避免了桥接和上下文切换。Flutter的热重载（hot-reload）特性亦让人惊叹，它能在编写代码时提供超快的视觉反馈，我们推荐你在项目中尝试使用Flutter。

jest-when
试验

jest-when是一个轻量级的JavaScript库，通过匹配模拟函数调用的参数完善了Jest的功能。Jest是测试整个技术栈的好工具，而jest-when可以帮助检查模拟函数接收的参数。这样就能够为具有很多依赖的模块写出更强壮的单元测试。

Micronaut
试验

Micronaut是一个JVM框架，可以用来构建Java，Kotlin或者Groovy服务。它没有通过运行时反射来完成依赖注入（DI）和生成代理（传统框架的常见缺点），而是使用了DI/AOP容器在编译时执行依赖注入，因此具有内存占用小、启动时间短的特点。这使得它不仅在标准的服务器端微服务方面，在物联网、Android应用程序和无服务器功能等环境中也很有吸引力。Micronaut使用Netty，并且对响应式编程提供一流的支持。它还包含了服务发现和熔断等特性，这些特性使得它对云计算非常友好。对于JVM领域的全栈框架来说，Micronaut是一个非常有前途的新成员。我们在越来越多的生产项目中看到了它的身影，这促使我们将其移至试用阶段。

采纳

试验
78. Arrow
79. Flutter
80. jest-when
81. Micronaut
82. React Hooks
83. React Testing Library
84. Styled components
85. Tensorflow

评估
86. Fairseq
87. Flair
88. Gatsby.js
89. GraphQL
90. KotlinTest
91. NestJS
92. Paged.js
93. Quarkus
94. SwiftUI
95. Testcontainers

暂缓
96. Enzyme

语言&框架

Micronaut是一个JVM框架，可以用来构建Java，Kotlin或者Groovy服务，具有内存占用小、启动时间短的特点。

（Micronaut）

在React前端方面，React Testing Library使其他选择黯然失色，成为明智的默认选择。

（React Testing Library）

React Hooks

试验

今年年初，React Hooks成为了流行的JavaScript框架。它无需编写类就可以使用状态和其他React功能，从而提供了一种比使用高阶组件或render-props更简洁的方法。诸如Material UI和Apollo之类的库已经切换到使用Hooks了。测试Hooks时会遇到一些问题，特别是使用Enzyme时，这能帮助我们重新评估是否选择Enzyme作为工具。

React Testing Library

试验

JavaScript世界日新月异，随着我们在框架使用方面的经验越来越多，我们的推荐也随之改变。有些框架随着我们的深入使用，会使其他类似框架都黯然失色。在React前端测试方面，React Testing Library就是这样一个例子。用它写的测试比其他框架（如Enzyme）脆弱，因为它鼓励独立测试组件间的关系，而不是测试全部实现细节。

Styled components

试验

使用带标签的模板文字styled components，可以将为React组件设置样式所需的CSS直接放入创建该组件的JavaScript代码中。这大大减轻了管理CSS的痛苦，并且不需要为避免CSS中的命名冲突而想尽办法，比如命名约定等。开发人员在查看组件定义时可以直接看到样式，而不必记住几MB的CSS样式。当然，将CSS放入JavaScript代码中，可能会使跨不同组件样式的一致性变得更加困难，因此我们建议使用这种方法时一定要理解其优缺点。

Tensorflow

试验

TensorFlow的2.0版本保持了其作为业界领先的机器学习框架的突出地位。TensorFlow最初是一个数字处理程序包，后来逐渐扩展为包括支持各种机器学习方法和执行环境（从移动CPU到大型GPU群集）的库。在此过程中，出现了许多框架，以简化网络创建和训练的任务。同时，其他框架（尤其是PyTorch）提供了一种命令式编程模型，该模型使调试和执行变得越来越容易。TensorFlow 2.0现在默认为命令流（立即执行），并采用Keras作为单个高阶API。尽管这些更改提高了TensorFlow的可用性并使其较PyTorch更具竞争力，但这是一次重大的重写，常常破坏向后兼容性——TensorFlow生态系统中的许多工具和服务框架都无法立即适配新版本。目前，请考虑是否要在TensorFlow 2.0中进行设计和试验，或恢复到版本1以在生产环境中服务和运行模型。

Fairseq

评估

Fairseq是Facebook AI Research的序列到序列建模工具套件，允许研究人员和开发人员训练定制模型以进行翻译、摘要、语言建模和其他NLP任务。对于PyTorch的用户来说，这是一个不错的选择。它提供了各种序列到序列模型的参考实现，支持跨多个GPU和机器的分布式训练，可扩展性强，并具有许多预训练的模型，其中包括RoBERTa，它是对BERT的优化。

Flair

评估

Flair是一个简单的基于Python的NLP框架。它让用户可以执行标准的NLP任务，例如命名实体识别（NER），词性标记（PoS），词义消歧和分类，并且在一系列NLP任务中都表现良好。Flair为各种文字和文档嵌入提供了一个简单且统一的界面，包括BERT、Elmo及其自己的Flair嵌入。同时，它还提供多语言支持。这个框架本身是建立在PyTorch之上的。我们在某些项目中正在使用它，并且喜欢它的易用性和强大的抽象。

Gatsby.js

评估

Gatsby.js是一个用于编写JAMstack架构风格网络应用的框架。应用的一部分在构建时生成并且以静态站点的形式进行部署。剩余的功能以渐进式网络应用的方式进行实现并运行在浏览器中。这些应用无需服务端代码即可运行。通常来说，渐进式网络应用会通过调用第三方API，或者现成的SaaS解决方案实现内容管理等功能。在Gatsby.js的例子中，所有的客户端和构建代码都是用React编写。框架包含了一些优化来让程序运行得更快。它将代码与数据分离来最大程度地减少加载时间，并且通过在应用内跳转时预先加载资源来提高性能。接口通过GraphQL进行调用并且通过一些插件来简化和现有服务的集成。

GraphQL

评估

我们已经在项目中看到了许多成功的GraphQL实现，也看到了一些有趣的模式和应用，比如将GraphQL用于服务器端资源聚合。尽管如此，对该框架的滥用、以及使用过程可能遇到的一些问题，我们并非毫无担忧。例如N+1查询可能带来的性能问题，以及添加新模型时需要大量样板代码而导致的复杂性等。这些问题有一些解决方法，例如使用查询缓存等。虽然这项技术不是银弹，我们仍然认为它值得作为系统架构的一部分进行评估。

KotlinTest

评估

在Kotlin生态系统中，KotlinTest是我们团队喜爱的独立测试工具。它提供了我们在之前的雷达中强调的技术——基于属性的测试。它的关键优势在于提供了多种测试方法以构建测试套件。同时，它内置了一组全面的匹配器，使我们能用优雅的内部DSL编写富有表现力的测试。

NestJS

评估

NestJS是使用TypeScript编写的服务器端框架。通过集成Node.js社区的丰富生态，NestJS提供了一种开箱即用的应用程序架构。开发NestJS的思维模型类似于Angular的服务器端版本或Spring Boot的TypeScript版本，因此开发人员的学习曲线很低。NestJS支持诸如GraphQL，Websocket和ORM库之类的协议。

Paged.js

评估

在使用HTML和相关技术来生产书籍和其他印刷品时，必须考虑分页问题。这包括页面计数器、页眉和页脚中的重复元素和分页符。Paged.js是一个开源代码库，它为Paged Media和Generated Content for Paged Media CSS模块实现了一系列补充代码。它仍处于试验阶段，但填补了HTML的“编写一次，到处发布”的重要空白。

Quarkus

评估

Quarkus是Red Hat的一个云原生、容器化优先的用于编写Java应用程序的框架。它具有非常快的启动时间（几十毫秒）和较低的内存占用率，这使其非常适用于Faas或者频繁的在容器编排中进行扩展和收缩。像Micronaut框架一样，Quarkus通过使用提前编译技术在编译时进行依赖注入，避免了反射造成的运行时成本。它还可以很好地和GraalVM的原生映像配合使用来进一步减少启动时间。Quarkus支持命令式和响应式模型。Quarkus与Micronaut和Helidon一起领导着新一代Java框架，这些框架试图在不牺牲开发人员效率的前提下，解决应用的启动性能和内存问题。它已经引起了社区的广泛兴趣，值得关注。

语言&框架

Gatsby.js是一个用于编写JAMstack架构风格网络应用的框架。它将代码与数据分离来最大程度地减少加载时间，并且通过在应用内跳转时预先加载资源来提高性能。

（Gatsby.js）

语言&框架

Quarkus是Red Hat的一个云原生、容器化优先的用于编写Java应用程序的框架。它具有非常快的启动时间（几十毫秒）和较低的内存占用率。

(Quarkus)

SwiftUI

评估

苹果在其新的SwiftUI框架上迈出了一大步，该框架用于在macOS和iOS平台上实现用户界面。我们很高兴SwiftUI跨越了Interface Builder和XCode之间略显混乱的关系，并采用了一致的、声明性的和以代码为中心的方式。现在，你可以在XCode 11中并排查看代码和生成的可视化界面，从而获得更好的开发人员体验。SwiftUI框架还从近年来主导Web开发的React.js的世界中汲取了灵感，它利用视图模型中的不可变值和异步更新机制，构成了统一的反应式编程模型。这为开发人员提供了一个完全原生的替代品，以替代类似React Native或Flutter之类的反应式框架。尽管SwiftUI确实代表了Apple UI开发的未来，但它是一个相当新的事物，还需要花费一些时间来打磨。我们期待改进的文档，和一个可以为测试与其他工程化问题建立一套实践的开发者社区。

Testcontainers

评估

随着现代系统的组件依赖数量不断增加，如何创建可靠的环境来运行自动化测试成为了一个反复出现的问题。Testcontainers是一个Java库，它通过在测试中施行容器化的依赖管理来缓解这一问题。这对于扩展可重复的数据库实例或类似的基础结构特别有用，同时它也可以在Web浏览器中用于UI测试。我们的团队发现利用此库提供的可编程、轻量且一次性的容器，集成测试会变得更加可靠。

Enzyme

暂缓

我们通常不会将已经移除的工具保留在技术雷达上，但是我们的团队强烈感受到Enzyme应该替换为React Testing Library来用于测试React界面组件。使用Enzyme的团队发现它对于被测试组件内部的聚焦会导致脆弱的、无法维护的测试。

想要了解技术雷达最新的新闻和洞见？

请选择你喜欢的渠道来关注我们

现在订阅

ThoughtWorks®

ThoughtWorks是一家软件咨询公司，也是一个充满热情、以目标为导向的社区。我们帮助客户以技术为核心，推动其商业变革，与他们并肩作战解决最核心的技术问题。我们致力于积极变革，希望能够通过软件技术创造更美好的社会，与此同时我们也与许多志向相投的组织合作。

创办25年以来，ThoughtWorks已经从一个小团队，成长为现在拥有超过7000人，分布于全球14个国家、拥有43间办公室的全球企业。这14个国家是：澳大利亚、巴西、加拿大、智利、中国、厄瓜多尔、德国、印度、意大利、新加坡、西班牙、泰国、英国、美国。

thoughtworks.com

ThoughtWorks®

thoughtworks.com/radar

#TWTechRadar